Modeling Sequence Data with RNNs, LSTMs

Simon Scheidegger
Department of Economics, University of Lausanne, Switzerland

March 26th, 2026 | University of Geneva

Road-Map

- This lecture:
  - Deep Learning cont'd
  - More advanced topics:
  - Recurrent neural networks and beyond.

Keras & Tensorflow Basics

- tensorflow.org
- Keras API: https://www.tensorflow.org/guide/keras/sequential_model
- Fun data sets to play with: https://www.kaggle.com/datasets
- Some ”clean” data to play with:
  https://archive.ics.uci.edu/ml/index.php
- Help for debugging — Tensorboard:
  https://www.tensorflow.org/tensorboard

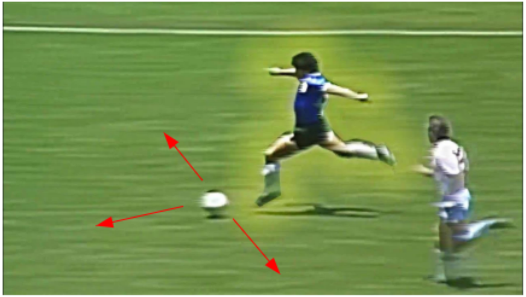
Beyond Vanilla DNN

- Not all applications are plain vanilla deep neural nets.
- There exist situations where more intricate architectures are needed.
- Examples:
  - Time-series comparisons, such as estimating how closely related two documents or two stock tickers are.
  - Sequence-to-sequence learning, such as decoding an English sentence into French.
  - Sentiment analysis, such as classifying the sentiment of tweets or movie reviews as positive or negative.
  - Time-series forecasting, such as predicting the future weather at a certain location, given recent weather data.

Example

Given a picture of a ball, can we predict where it will go?

Example



Given a picture of a ball, can we predict where it will go?

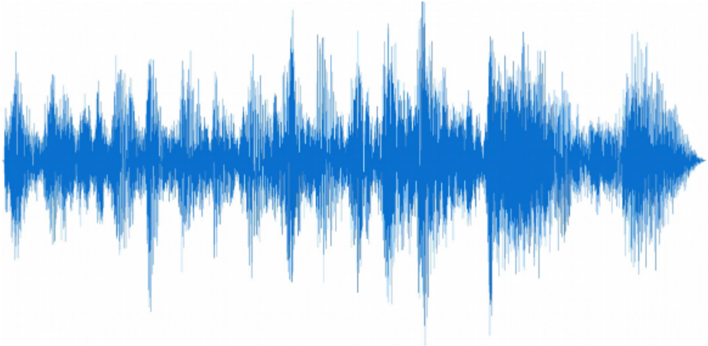
A Sequence Modeling Problem: Predict the Next Word

"Today, we are having a class on deep ______"

A Sequence Modeling Problem: Predict the Next Word

"Today, we are having a class on deep learning"

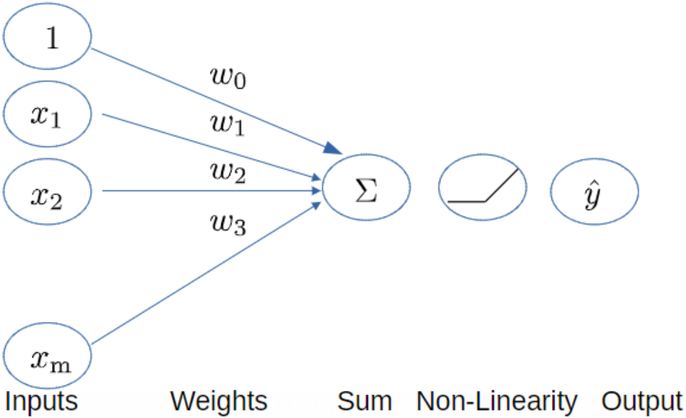
A Sequence Modeling Problem: Audio



A Sequence Modeling Problem: Beethoven

https://www.youtube.com/watch?v=Rvj30blscqw

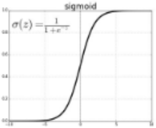
The Perceptron Revisited



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

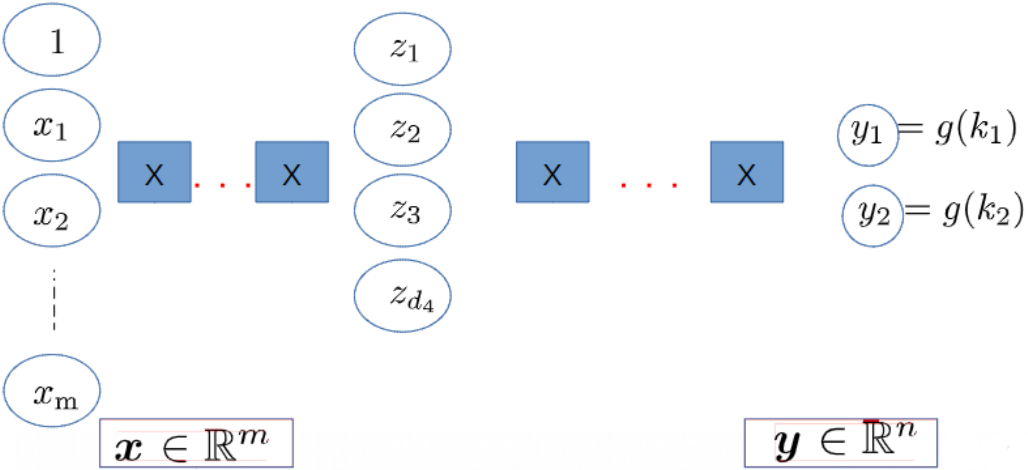
Activation Functions
e.g. sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$



$\Longrightarrow$ Bias term allows you to shift your activation function to the left or the right

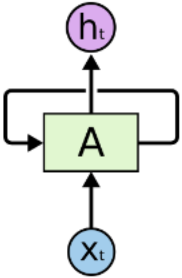
Feed-Forward Nets Revisited



Recurrent Neural Nets

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- To model sequences, we need to
  - Handle variable-length sequences.
  - Track long-term dependencies.
  - Maintain information about the order.
  - Share parameters across the sequence.



- Recurrent Neural Networks (RNN) are an approach to sequence modeling problems (Rumelhart et al. (1986)).
- More specifically, given an observation sequence $x = \{x_1, x_2, \ldots, x_T\}$ and its corresponding label $y = \{y_1, y_2, \ldots, y_T\}$, we want to learn a map $f : x \to y$.

RNN

- RNNs are a family of neural networks for processing sequential data.
- A RNN is a neural network that is specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$.
- Unfold the computational graph of a dynamical system:

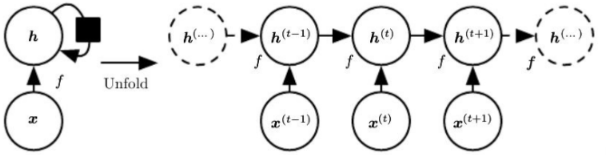
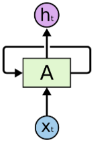


Fig. from Goodfellow et al. (2016)

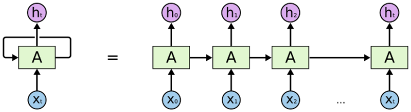
Preview on RNN

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Recurrent layers use their own output as input.
  - In Figure: A is a recurrent cell
- Introduce history or time dependency in NNs.
- The only way to efficiently train them is to unroll them.

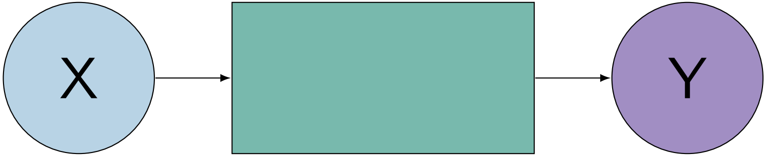


$$\underset{\text{Cell state}}{\boxed{\boldsymbol{h}^{(t)}}} = \underset{\text{Function (parameterized)}}{\boxed{f}} \left(\underset{\text{Old state}}{\boxed{\boldsymbol{h}^{(t-1)}}} , \underset{\text{Input vector at time } t}{\boxed{\boldsymbol{x}^{(t)}}} ; \boldsymbol{\theta} \right)$$



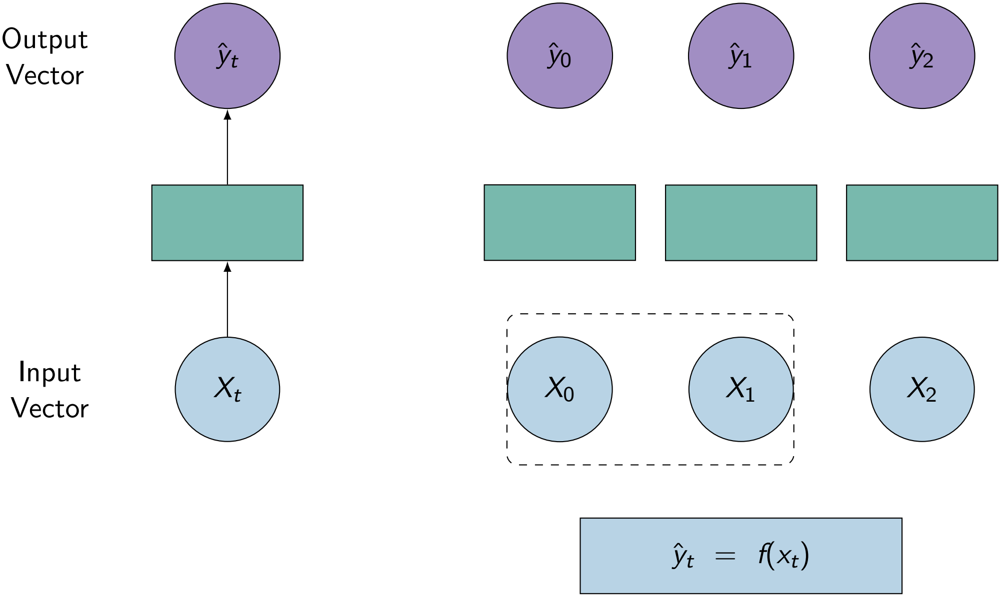
An unrolled recurrent neural network.

Feed-Forward Nets Revisited

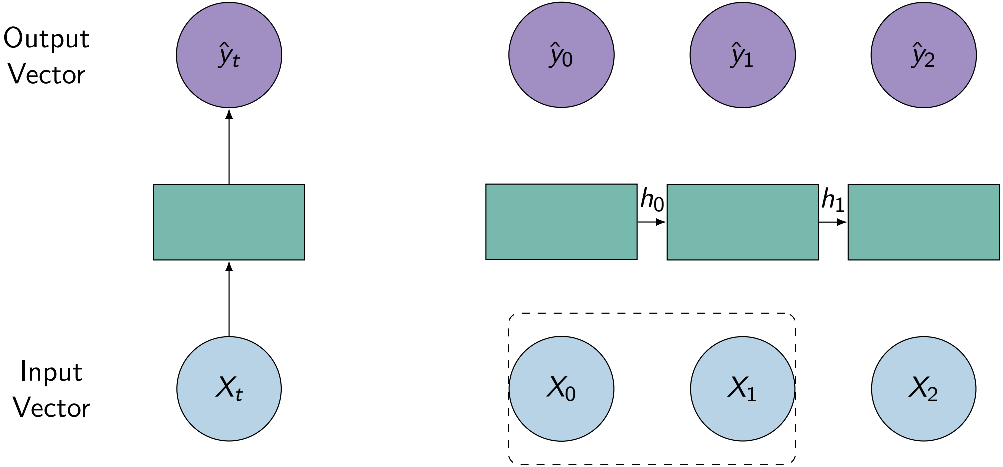


$x \in \mathbb{R}^m$ $\quad$ $y \in \mathbb{R}^n$

Handling Individual Time Steps

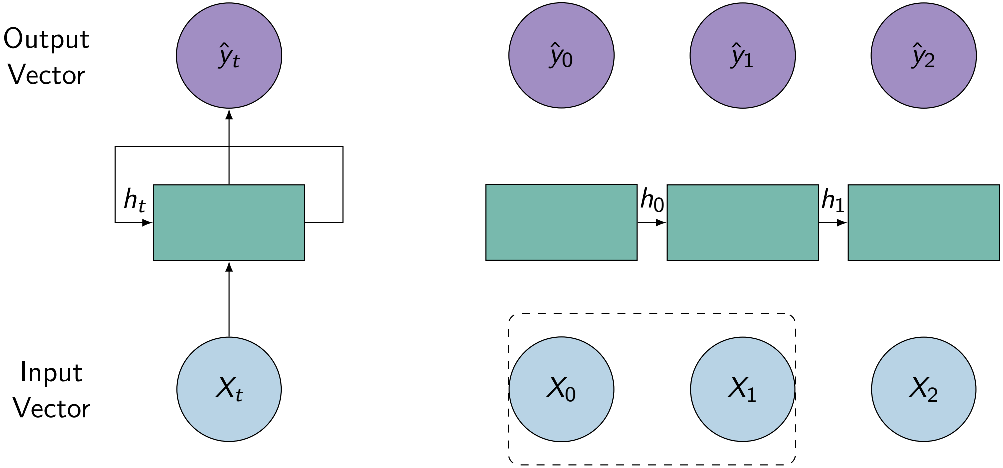


Neurons With Recurrence



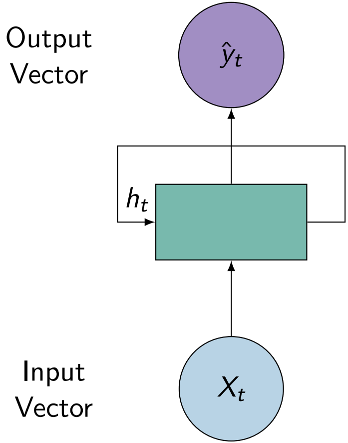
$$\underbrace{\hat{y}_t}_{\text{output}} = f\,(\underbrace{x_t}_{\text{input}},\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Neurons With Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f\,(\underbrace{x_t}_{\text{input}}\,,\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks



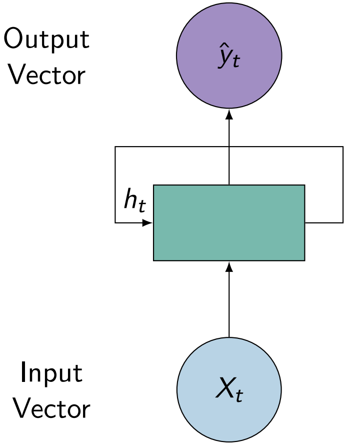
- Apply a recurrence relation at every time step to process a sequence:

$$\underbrace{\boldsymbol{h}_t}_{\text{cell state}} = \underbrace{f_W}_{\text{function with weights W}} \left(\underbrace{\boldsymbol{x}_t}_{\text{input}}, \underbrace{\boldsymbol{h}_{t-1}}_{\text{old state}} \right)$$

- cell state function input old state with weights W
- Note: the same function and set of parameters are used at every time step.

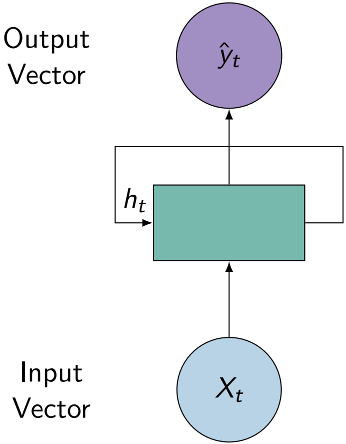
RNNs have a **cell state** that is updated **at each time step** as a sequence is proceeded.

RNN Intuition



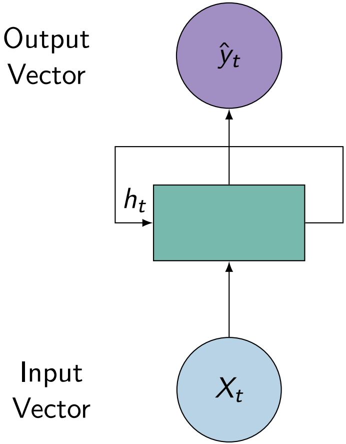

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN Intuition



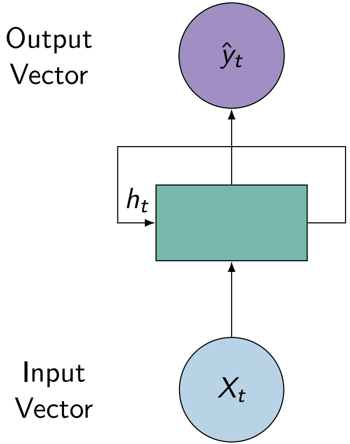

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN Intuition




```python
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN State Update And Output



- Output vector

$$\hat{\boldsymbol{y}}_t = W_{hy}^T h_t$$

- Update hidden state

$$\mathrm{h}_t = \tanh\left(\boldsymbol{W_{hh}^T} h_{t-1} + \boldsymbol{W_{xh}^T} x_t\right)$$

- Input vector

$$x_t$$

RNN In One Slide

- RNN models a dynamic system, where the hidden (cell) state h_t is not only dependent on the current observation x, but also relies on the previous hidden state h.
- More specifically, we can represent h_t as $\mathbf{h}_t = \mathbf{f}\left(\mathbf{h}_{t\text{-}1}, \mathbf{x}_t\right)$ (Eq. 1) where f is a nonlinear (time-invariant) mapping.
- Thus, h_t contains information about the whole sequence, which can be inferred from the recursive definition in $\mathrm{Eq.1}$.
- In other words, RNN can use the hidden variables as a memory to capture long term information from Figare 1: It is a RNN example corresponding extended RNN moil a sequence.
- Prediction at the time step $t : z_t$

$$\mathbf{h}_t = \tanh\left(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t + \mathbf{b_h}\right)$$

$$z_t = \operatorname{softmax}\left(W_{hz}\mathbf{h}_t + \mathbf{b}_z\right)$$

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum y_t \log z_t$$

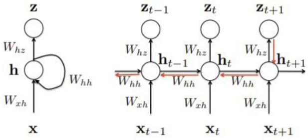
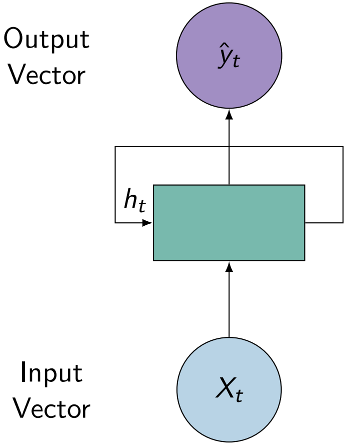


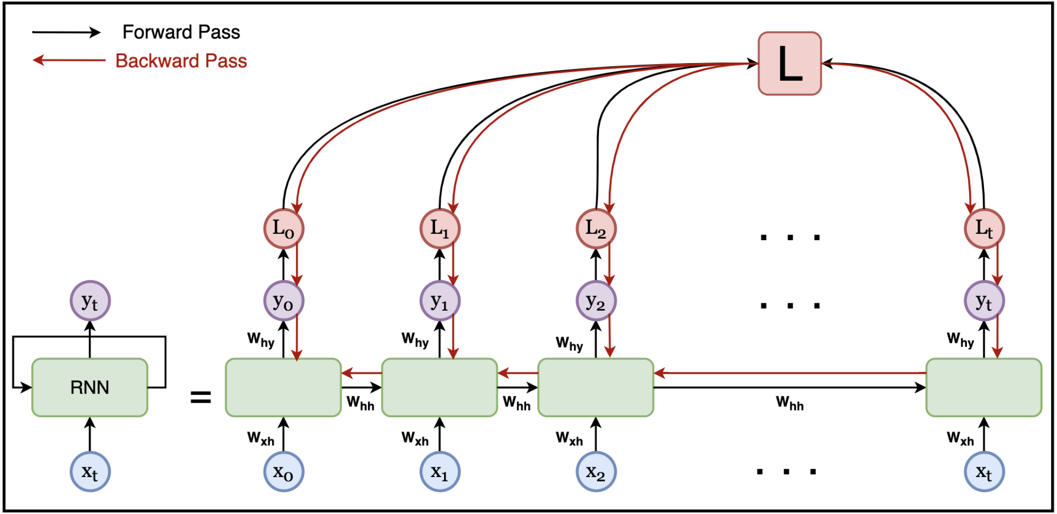
Figure 1: It is a RNN example: the left recursive description for RNNs, and the right is the corresponding extended RNN model in a time sequential manner.

RNN: Computation Graph Across Time



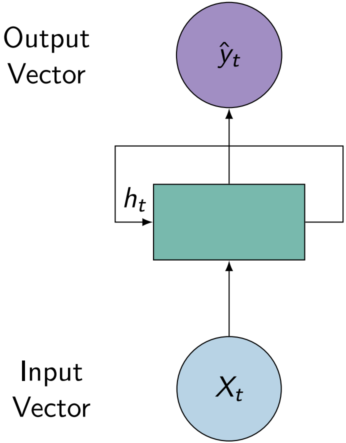
$\rightarrow$ represent as computational graph unrolled across time.

Back-Propagation Through Time



Re-use same weight matrices at every time step!

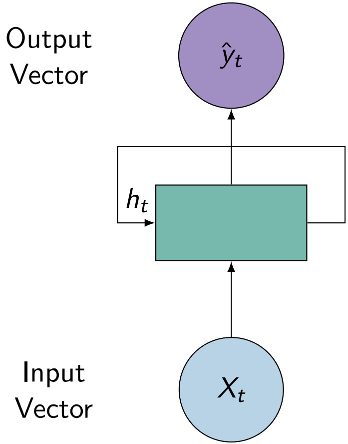
RNN From Scratch & Tensorflow

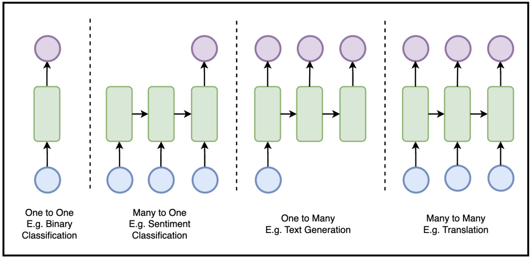



```python
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```

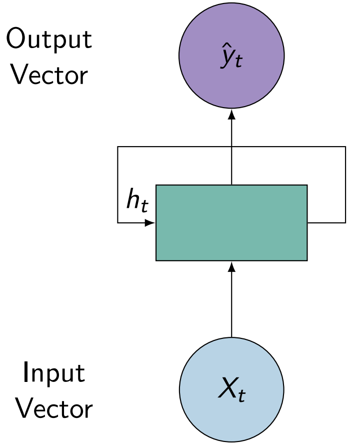
```python
tf.keras.layers.SimpleRNN(rnn_units)
```

RNN Intuition





Sequence Modeling — Design Criteria

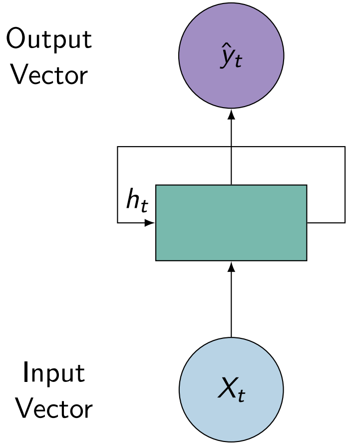


Recall: to model sequences, we need to:

- Handle variable-length sequences.
- Track long-term dependencies.
- Maintain information about order.
- Share parameters across the sequence.

$\rightarrow$ Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria.

Handle Variable Sequence Lengths



The food was great.

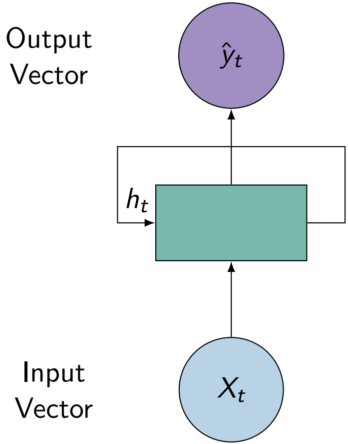
vs.

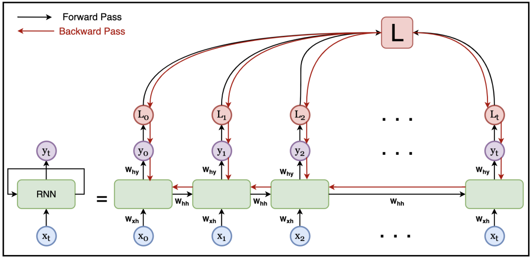
We visited a Pizzeria for lunch.

vs.

We were hungry because we went for sport before eating.

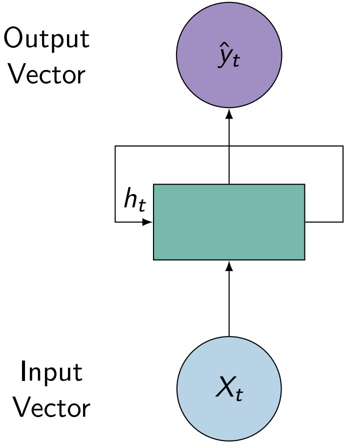
Backpropagation Through Time

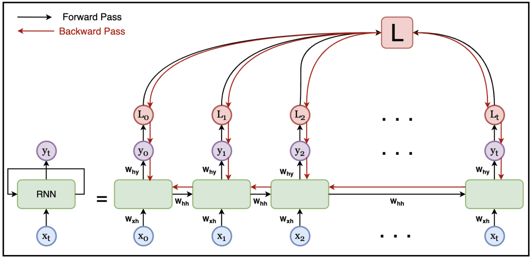




Computing the gradient wrt. h_0 involves many factors of W_{hh}+ repeated gradient computation!

Backpropagation Through Time





Many values > 1:
Exploding gradients

Many values < 1:
Vanishing gradients

RNNs with PyTorch

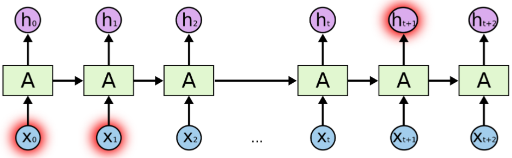
Action required:
$\rightarrow /lectures/day5/code/rnn_lstm_tutorial.ipynb$

Recall: RNN Hard to Train

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent blocks suffer from two problems:

- Long-term dependencies do not work well.
  - Difficult to connect two distant parts of the input.
- Magnitude of the signal can get amplified at each recurrent connection.
  - At every time iteration, the gradient can either vanish or explode.
  - Very hard to train them.

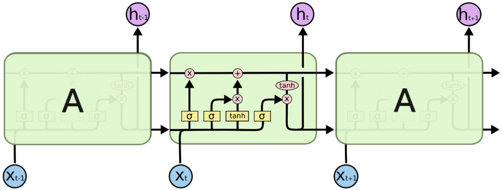


I grew up in England... and I speak fluent ____

Long Short-Term Memory (LSTM)

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Hochreiter & Schmidhuber (1997)
- LSTM layers are improved versions of the recurrent layers.
  - They rely on a gated cell to track information throughout many time steps.
  - They can learn long-term dependencies.
  - They can forget.
- They have an internal state and a structure which is composed of four actual layers.
  - Layers labeled with σ are gates which can block or let information flow.



Long Short-Term Memory (LSTM)

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The core of LSTM is a memory unit (or cell) c_t which encodes the information of the inputs that have been observed up to that step.
- The memory cell c_t has the same inputs ($\mathbf{h}_{t-1}$ and $\mathbf{x}_t$) and outputs $\mathbf{h}_t$ as a normal recurrent network, but has more gating units which control the information flow.
- The input gate and output gate respectively control the information input to the memory unit and the information output from the unit. More specifically, the output h_t of the LSTM cell can be shut off via the output gate.

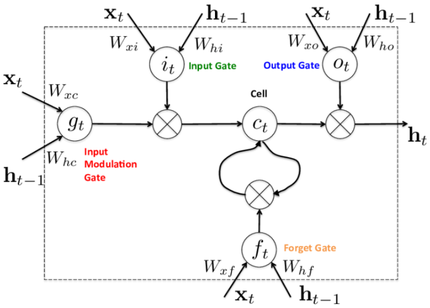
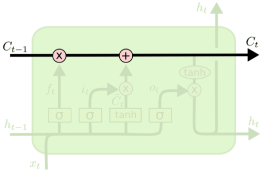


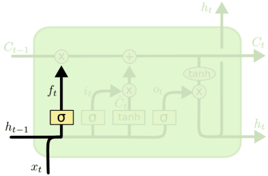
Fig. from G. Chen (2016)

LSTM Forget Gate

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- LSTMs follow two paths
  - They update their internal state.
  - They give an output based on the internal state and the input.
- A gate layer σ decides if we should forget an old part of the internal state.
- Something which has to be replaced by new information.



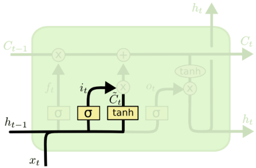


1. **Forget**
2. Store
3. Update
4. Output

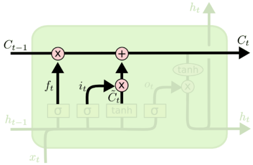
LSTM New State

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Once that the layer decided what to forget, it computes:
  - What has to replace it, i, based on the input and the old state.
  - What has to be used to replace it, the candidate value C_t
- The new state C_t can be computed based on the new information.



1. Forget
2. **Store**
3. Update
4. Output

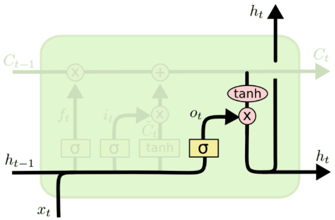


1. Forget
2. Store
3. **Update**
4. Output

LSTM Output

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Based on the new state and the input, the layer can produce a result.
  - this is the output.
  - the same value is also passed to the next iteration.
- Why is this so important?
  - Many translation algorithms and voice interpreters are based on small variations of this layer.
- Action required: /lectures/day5/code/RNN_intro.ipynb (see also https://www.tensorflow.org/guide/keras/rnn)



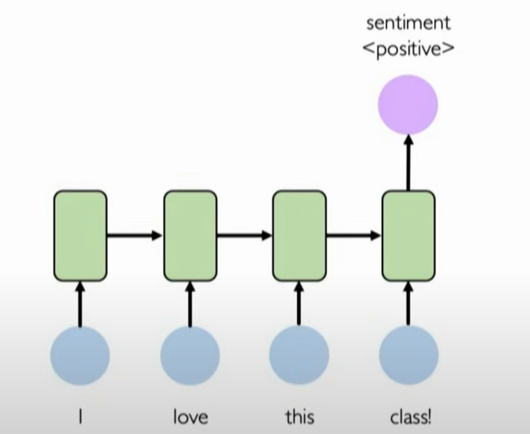
1. Forget
2. Store
3. Update
4. **Output**

Action Required

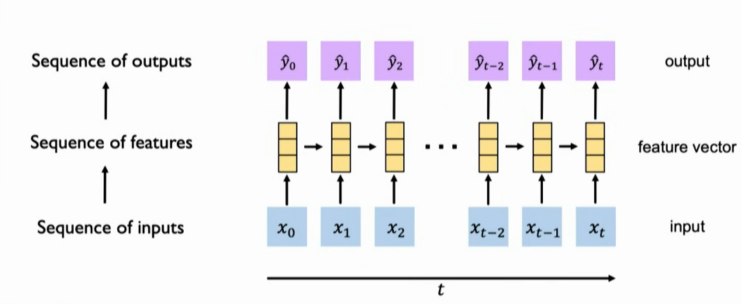
- `/lectures/day5/code/RNN_intro.ipynb` (Ozone and stock market time series data).
- There is a weather data set from the Max Planck Institute of Biochemistry `https://www.bgc-jena.mpg.de/wetter/`.
- Open the notebook `demo/05b_Weather_data.ipynb`.
- Given this time series (Temperature as a function of time), try to make predictions of various time intervals into the future.

Limitations of Recurrent Models

- Encoding bottleneck
- Slow, no parallelization
- Not long memory (very long sequences cannot be handled, not even by LSTMs)
- Can we go beyond those limitations to process sequential data?



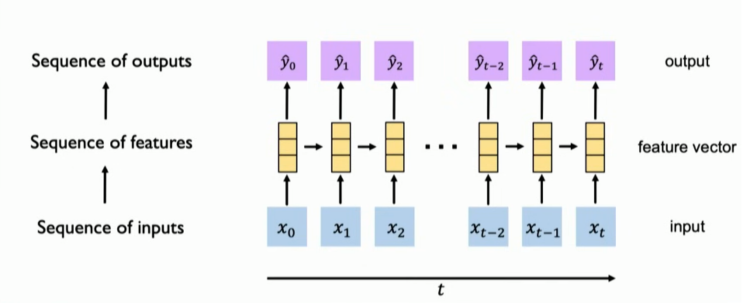
High-level Goal of Sequence Modeling



RNNs: recurrence to model sequence dependencies

- Encoding bottleneck
- Slow, no parallelization
- Not long memory (very long sequences cannot be handled, not even by LSTMs)
- Can we go beyond those limitations to process sequential data?

Desired Capabilities



RNNs: recurrence to model sequence dependencies

- Continuous stream
- Parallelization
- Long memory

→ Can we eliminate the need for recurrence entirely (i.e., no need to process the data time-step by time-step)?

Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for sequence modeling tasks.
2. Model sequences via a recurrence relation.
3. Training RNNs with backpropagation through time.

Questions?